VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÝ NÁSTROJ PRO SPRÁVU VÝSLEDKŮ DLOUHODOBÉHO TESTOVÁNÍ

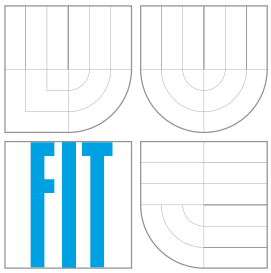
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

FILIP MATYS

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÝ NÁSTROJ PRO SPRÁVU VÝSLEDKŮ DLOUHODOBÉHO TESTOVÁNÍ

WEB TOOL FOR MANAGEMENT OF LONG-TERM TESTING RESULTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — FILIP MATYS
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR MÜLLER
SUPERVISOR

BRNO 2014

Abstrakt

Mnoho softwarových projektů prochází dlouhým vývojem, proto vyžadují rozsáhlé testovací sady pro automatizaci testování. Tyto testovací sady často neprochází celé a je potřeba jejich výsledky s časem shromažďovat a následně analyzovat, jak se daná testovací sada chovala dříve. V rámci bakalářské práce byl implementován webový nástroj, který umožňuje zpracování výsledků testovacích sad, které zařazuje do kontextu vývoje a umožňuje jejich analýzu. Díky své modularitě není závislý na konkrétním formátu výsledků. V práci jsou použity moderní nástroje pro tvorbu uživatelského rozhraní, a to AngularJS a Foundation, pomocí kterých bylo docíleno rychlejší odezvy systému a schopnosti přizpůsobit se obrazovkám mobilních zařízení. Pro demonstraci možností nástroje byly implementovány tři zásuvné moduly různých formátů výsledků testovacích sad. Výsledný nástroj byl nasazen na službu OpenShift, kde byla importována data z výsledkových sad SystemTap a glibc testsuite a na dvou případových studiích byl prezentován jeho přínos.

Abstract

Many software projects go through a long evolution, which results in creation of extensive test suites to automate testing process. These test suites often do not pass whole, so it is needed to collect their results and then analyse them, how did the test suite behave before. Within this thesis, a web tool was implemented, which allows to process results of test suites that are afterwards set in the context of development and allows their analysis. Thanks to it's modularity, the tool is not dependent on results format. The tool uses modern frameworks, such as AngularJS and Foundation, to aim at lower system latency and to adapt it to screens of mobile devices. To demonstrate the tool, three plug-ins were implemented for different formats of test suite results. The tool was deployed on the OpenShift service. There were imported data from SystemTap and glibc testsuite result sets and benefits of the tools were presented in two case studies.

Klíčová slova

webový nástroj, správa, testování, testovací sada, PHP, JavaScript, MySQL

Keywords

web tool, management, test suite, PHP, JavaScript, MySQL

Citace

Filip Matys: Webový nástroj pro správu výsledků dlouhodobého testování, bakalářská práce, Brno, FIT VUT v Brně, 2014

Webový nástroj pro správu výsledků dlouhodobého testování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Müllera.

.....
Filip Matys
21. května 2014

Poděkování

Děkuji svému vedoucímu, panu Ing. Petru Müllerovi, za vedení a pomoc během tvorby této bakalářské práce, svým přátelům, kteří dokázali v těžkých chvílích podržet, a své rodině, která mi poskytuje zázemí.

Obsah

Kapitola 1

Úvod

Tak, jako je potřeba pravidelně kontrolovat zdraví člověka, je potřeba testovat software během jeho životního cyklu. Na počátku se jedná o ladění chyb a kontrolu, zda vše probíhá tak, jak bylo předem specifikováno, později je sledován dopad záplat a vylepšení. Takovým softwarem je například Linuxové jádro, jehož vývoj probíhá od roku 1991 a trvá dodnes. Podílí se na něm téměř 1000 vývojářů ze 100 různých firem a denně se jádro doplní průměrně o necelé čtyři tisíce řádků. Takto komplexní vývoj je třeba pravidelně testovat a analyzovat výsledky, aby nedošlo k zablokování funkčních celků či k naprostému zastavení funkcionality jádra.

Aby byl tento proces ulehčen, vznikají testovací sady, které produkují množství dat, jež je potřeba zpracovat a zobrazit v pro člověka čitelné formě. Nástroje, které tento problém řeší, jako například Dejazilla [3], jsou již zastaralé a zaostávají svou implementací za vývojem samotných aplikací. Nevýhodou těchto nástrojů je nemožnost rozšíření o další formáty testovacích sad a s narůstajícím podílem mobilních zařízení s přístupem na internet i neschopnost prezentovat data v responzivní podobě. Cílem této bakalářské práce bylo vytvořit nástroj, který bude schopen tyto nedostatky odstranit, a zároveň bude schopen zpracovat více formátů testovacích sad.

Tento text je rozdělen do pěti kapitol. Kapitola následující seznamuje čtenáře se specifiky dlouhodobého testování. Další dvě kapitoly se zabývají návrhem a následnou implementací aplikace. Předposlední kapitola zpracovává případové studie a poslední shrnuje dosažené výsledky a další vývoj.

Kapitola 2

Specifika dlouhodobého testování

Pro sledování kvality a chování aplikací, či jejich jednotlivých částí, slouží proces testování. Tento proces může být prováděn buď manuálně, nebo může být zautomatizován pomocí různých nástrojů. Tyto nástroje vytváří textové výstupy, které jsou mnohdy velice rozsáhlé a v surové formě těžko čitelné. Je proto nutné mít nástroj pro jejich zpracování a zobrazení výsledků v jednoznačné čitelné podobě.

2.1 Pojmy

Testovací případ Testovacím případem je myšlen scénář se vstupem, výstupem, konkrétním výsledkem a výsledkem aktuálně testovaného programu, který je v rámci testovací sady jednoznačně identifikován. Takovýto scénář se může skládat z jednoho či více kroků, které se mají během konkrétního testování provést.

Testovací sada Soubor testovacích případů tvoří testovací sadu, která má za úkol otestovat konkrétní chování programu či programové části. Mnohdy důležitou součástí testovací sady je i identifikace systémové konfigurace použité během testování. Vytvořená testovací sada je opakovaně a tedy i dlouhodobě využívána pro sledování různých dopadů na systém.

Softwarová regrese a regresní testování Softwarová regrese je stav, kdy dojde ke zhoršení funkcionality aplikace po určitých změnách, kterými mohou být softwarové záplaty, změny konfigurace nebo přidání nových funkcí. Zhoršení se může projevit několika způsoby. Přestane fungovat jistá programová část, která předtím fungovala, dojde ke snížení bezpečnosti systému, či se zhorší celkový výkon. Regresní testování má za úkol takovéto stavy odhalit. Speciálním typem regresního testování je regresní benchmarking, během kterého dochází k testování výkonu software v pravidelných intervalech [1].

2.2 Proces dlouhodobého testování

Velké množství softwarových projektů je vyvíjeno velmi dlouho a je za potřebí je během tohoto vývoje udržovat. K tomuto slouží rozsáhlé testovací sady. Průběh testovací sady však nemusí být bez chyb, je proto nutné jejich výsledky shromažďovat v kontextu vývoje a následné provádět jejich analýzu, jakým způsobem se výsledky dané testovací sady mění v průběhu vývoje.

Pro popis procesu dlouhodobého testování byl zvolen nástroj SystemTap. Tento nástroj je využíván pro sledování běhu linuxových distribucí, pro které umožňuje sběr dat s minimální režií napříč celým systémem, ať už se jedná o jádro nebo uživatelský prostor [10].

2.2.1 Popis nástroje SystemTap

SystemTap definuje skriptovací jazyk *stap*, pomocí kterého uživatel definuje sondy a akce. Tyto sondy mají za úkol zachytávat určité uživatelem definované události. Těmi mohou být vstup do funkce, výstup z funkce, vypršení časovače, ukončení sezení a podobně. Po zachycení události dochází k provedení série kroků definované uživatelem právě skriptovacím jazykem *stap*. Zpravidla se jedná o získání dat, jejich uložení či tisk.

Soubory vytvořené uživatelem pomocí nástroje SystemTap jsou ukládány s příponou .stp a spouštěny skrze příkazovou řádku. Před spuštěním, respektive zavedením všech definovaných sond do systému, jsou soubory systémem zkontrolovány, načež jsou následně zkompilovány do kódu rozšiřujícího běžící jádro kernel, takzvaného *loadable kernel module*. Po ukončení nástroje je modul z jádra zase uvolněn.

2.2.2 Testovací sada nástroje SystemTap

Testovací sada nástroje SystemTap je spouštěna pomocí testovacího rozhraní DejaGnu [9]. Toto rozhraní poskytuje jednotný front-end pro všechna testování. Je součástí GNU Projektu a založen na nástroji pro automatizaci Expect, který využívá skriptovací jazyk Tcl. DejaGnu pomocí jednoho hlavního skriptu projde adresář s konfiguračními soubory testovací sady a následně začne provádět definované akce, respektive testy.

Testovací sada nástroje SystemTap se skládá z jednoho či více skriptovacích souborů. V těchto souborech jsou definovány sondy pro zachytávání událostí a akce, které se po zachycení události mají provést. Každá sonda, respektive událost, představuje jeden testovací případ. Událost může mít nadefinováno více akcí, může se tedy v rámci testovacího případu objevit více výsledků, jejichž počet není předem určený.

2.2.3 Příklad výstupních dat testovací sady nástroje SystemTap

Na obrázku 2.1 je zobrazena ukázka jednoho spuštění testovací sady nástroje SystemTap, která byla poskytnuta vedoucím práce. Kompletní výsledky různých testovacích sad lze nalézt na webových stránkách nástroje [2].

Uživatelem vytvořené soubory pomocí nástroje SystemTap, využívané pro zachytávání informací o běžící linuxové distribuci, a které jsou kompilovány do jednoho kernel modulu, tvoří jednu testovací sadu. Tato testovací sada je dělena do kategorií definovaných jednotlivými soubory. Tyto jednotlivé kategorie pak obsahují testovací případy, které jsou zastupovány událostmi. V ukázce je lze rozpoznat pomocí výrazu `Running`, přípona slova `systemtap` pak identifikuje kategorii. Výstupy kroků, ke kterým dochází po zachycení události, pak tvoří výsledky těchto případů. Jednotlivé výsledky jsou pak předmětem zpracování různých nástrojů. Na ukázce lze rozpoznat i výpis konfigurace, na které byl nástroj spuštěn.

```
Test Run By root on Mon Aug 19 13:43:13 2013
Native configuration is x86_64-unknown-linux-gnu

=== systemtap tests ===

Schedule of variations:
    unix

Running target unix
Using /usr/share/dejagnu/baseboards/unix.exp as board description file for
target.
Using /usr/share/dejagnu/config/unix.exp as generic interface file for target.
Using ./config/unix.exp as tool-and-target-specific interface file.

Host: Linux pes-guest-104.lab.eng.brq.redhat.com 3.10.0-0.rc7.64.el7.x86_64
 #1 SMP Tue Jun 25 10:00:04 EDT 2013 x86_64 x86_64 x86_64 GNU/Linux
Snapshot: version 2.2.1/0.155, rpm 2.2.1-1.el7
GCC: 4.8.1 [gcc (GCC) 4.8.1 20130612 (Red Hat 4.8.1-2)]
Distro: Red Hat Enterprise Linux Server release 7.0 Beta (Maipo)
SElinux: Enforcing


Running ./systemtap/notest.exp ...
testcase ./systemtap/notest.exp completed in 0 seconds
Running ./systemtap.apps/java.exp ...
UNTESTED: java - no javac
```

Obrázek 2.1: Ukázka výstupních dat nástroje SystemTap

Jedna kompilace modulu může být spuštěna pouze na systému, na kterém byla zkompilována. Pro použití testovací sady na jiných systémech je potřeba ji na daném systému zkompilovat. Tento krok však může být v některých případech problémem, například pokud daný systém kompilaci neumožňuje. Řešením tohoto problému je *cross-instrumentation*, popsaná v dokumentaci společnosti Red Hat [8]. Výsledky testovací sady z různých systémů jsou na sobě nezávislé, tudíž tvoří samostatné testovací cykly.

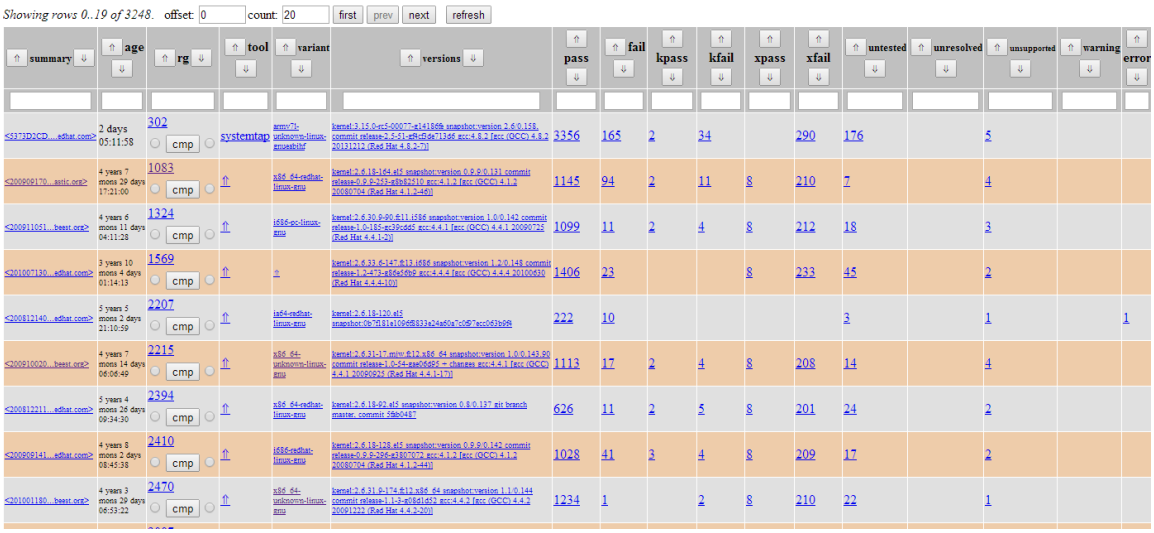
Každé spuštění testovací sady je identifikováno časovým razítkem, které je umístěno na prvním řádku výpisu. Kromě identifikace tato informace dále umožňuje jednotlivá spuštění řadit podle času.

2.2.4 Zpracování dat nástroje SystemTap

Výsledek jednoho průběhu testovací sady nástroje SystemTap je reprezentován textovým souborem. Jednoduchý výpis výsledků v jejich textové podobě lze nalézt na wiki stránkách nástroje [2]. Pro komplexnější zpracování výsledků a jejich zobrazení existuje nástroj nazvaný Dejazilla [3], jehož screenshot je na obrázku 2.2.

Dejazilla Jedná se o webový nástroj pro správu výsledků dlouhodobého testování, který zpracovává data různých formátů testovacích sad, zejména je však využíván právě pro Sys-

temTap. Data přijímá skrze elektronickou poštu, přičemž obsah zprávy je brán jako výstup testovací sady. Tento nástroj je schopen zobrazit souhrnný výpis, či seznam neproběhlých testů.



Obrázek 2.2: Dejazilla

Výstupy tento nástroj zobrazuje jako stránkovaná tabulková data, kde lze sloupce filtrovat pomocí SQL dotazů. Uživatelský pohled je rozdělen na tři části, a to na výpis výsledkových sad, detail výsledkové sady a rozdíl dvou výsledkových sad.

Výpis výsledkových sad Tento pohled zobrazuje výpis výsledkových sad s počty jednotlivých výsledků, dále pak označení, stáří, použitý nástroj či označení varianty. Pomocí tohoto pohledu se lze dostat na detail výsledkové sady nebo zobrazit rozdíl dvou výsledkových sad.

Detail výsledkové sady V detailu jsou vypsány veškeré výsledky zvolené sady. Ke každému výsledku je uveden i testovací případ, ke kterému se vztahuje. Nevýhodou tohoto pohledu je, že se zde objevují redundantní data, například označení sady a její stáří. Seskupení výsledků podle testovacího případu lze dosáhnout seřazením dat podle testovacího případu. K rozlišení charakteru hodnot jsou tyto hodnoty barevně podbarveny, což umožňuje snadnější uživatelskou detekci záporných výsledků.

Rozdíl dvou výsledkových sad Rozdíl dvou výsledkových sad zobrazuje rozdíl výsledků podle testovacích případů, které jsou v pohledu sjednoceny. Výsledky každé výsledkové sady jsou pak rozděleny do samostatných sloupců, jsou tedy podle testovacího případu postaveny vedle sebe. Nevýhodou je, že přechod do výsledku rozdílné hodnoty zde není nijak vyznačen, barevná podbarvení hodnot odpovídají standardnímu výpisu výsledků z detailu výsledkové sady. Dále pohled umožňuje rozdíl pouze dvou výsledkových sad.

Dejazilla představuje komplexní nástroj, jehož největší síla je ve filtrování dat pomocí jazyka SQL a jejich řazení. Data zobrazuje ve formě tabulek, přičemž v některých zobrazeních dochází k redundanci určitých dat. Hlavní nevýhodou nástroje je, že je schopen

zpracovávat pouze omezené množství formátů testovacích sad. Další nevýhody se týkají zejména pohledu pro rozdíl dvou výsledkových sad, který neumožňuje porovnání více jak dvou výsledkových sad a zároveň žádným způsobem nezvýrazňuje přechody do rozdílného výsledku. Nevýhodou také je, že veškerou funkcionalitu, respektive zobrazení dat, zajišťuje jedna komponenta, která pro všechny formáty testovacích sad zobrazuje stejný formát výstupu.

Kapitola 3

Návrh aplikace

Návrh a implementace výsledné aplikace vychází z poznatků o dlouhodobém testování popsaných v kapitole 2. Cílem je vytvořit nástroj, který bude schopen rozšiřitelnosti o různé formáty testovacích sad a komponent pro zobrazení výsledků jejich zpracování. Tato funkcionalita je zajišťována zásuvnými moduly, o které bude možno aplikaci rozšířit bez zásahu do jejího výpočetního jádra. Tato kapitola popisuje obecné rozdělení aplikace na logické celky, které zajištují klíčové úkony pro splnění na ni kladených požadavků. Řešení konkrétních problémů a popis použitých nástrojů se nachází v kapitole 4.

3.1 Architektura

Logickými celky aplikace jsou v tomto případě tři vrstvy. Datová vrstva, aplikační vrstva a prezentační vrstva. Toto rozdělení odpovídá vzoru Model-View-Controller, tedy MVC, kdy jednotlivé vrstvy tvoří nezávislé komponenty, které mezi sebou komunikují. Tato architektura byla zvolena z důvodu jednodušší správy jednotlivých vrstev, dále pak je možno díky tomuto přístupu využít implementační nástroje nezávislé na jednotlivých vrstvách. Je však nutno dbát na jednotné rozhraní, aby nedošlo k narušení vzájemné komunikace.

3.1.1 Model

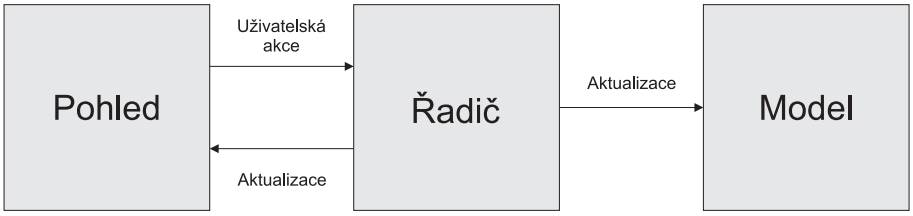
Datová vrstva je reprezentována modelem. Jeho úkolem je uchovávat data, zachovávat jejich integritu a být schopen je poskytovat aplikační vrstvě, od které mimo jiné přijímá data nová, či požadavky na jejich modifikaci. Zpravidla je tato vrstva reprezentována databází.

3.1.2 Pohled (View)

Zobrazení výstupu aplikace a možnosti interakce zajištuje Pohled. Tato vrstva přijímá požadavky uživatele, které přeposílá řadiči, jež tyto požadavky zpracovává a řídí další průběh aplikace. Mimo požadavků může Pohled reagovat i na procesy, které mohou vzniknout na základě jiných okolností, daných například časem, či samotnou inicializací konkrétní scény.

3.1.3 Řadič (Controller)

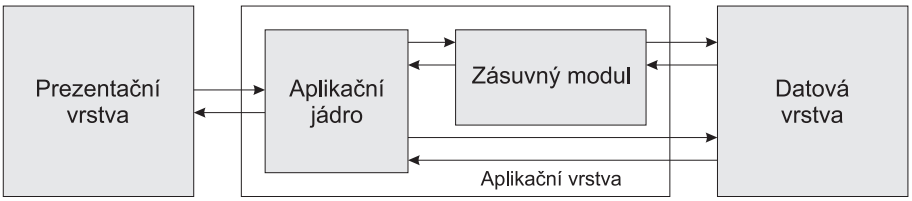
Hlavní výpočetní komponentou a spojnicí mezi prezentační a datovou vrstvou je Řadič. Tato vrstva zajištuje aplikační logiku, tedy transformaci dat a jejich proudění mezi vrstvami. V tomto konkrétním případě dále musí zajistit komunikaci se zásuvnými moduly pomocí obecného rozhraní.



Obrázek 3.1: Architektura Model-View-Controller

3.2 Aplikační struktura

Výsledný nástroj je navržen jako webová aplikace, která využívá architekturu MVC. Aby však bylo možno aplikaci libovolně rozšiřovat o nové formáty testovacích sad, je nutné tomuto nástroji umožnit rozšiřitelnost. Této vlastnosti je dosaženo pomocí zásuvných modulů, o které je možné nástroj rozšiřovat. Na obrázku 3.2 je schéma architektury MVC doplněné o zásuvný modul.



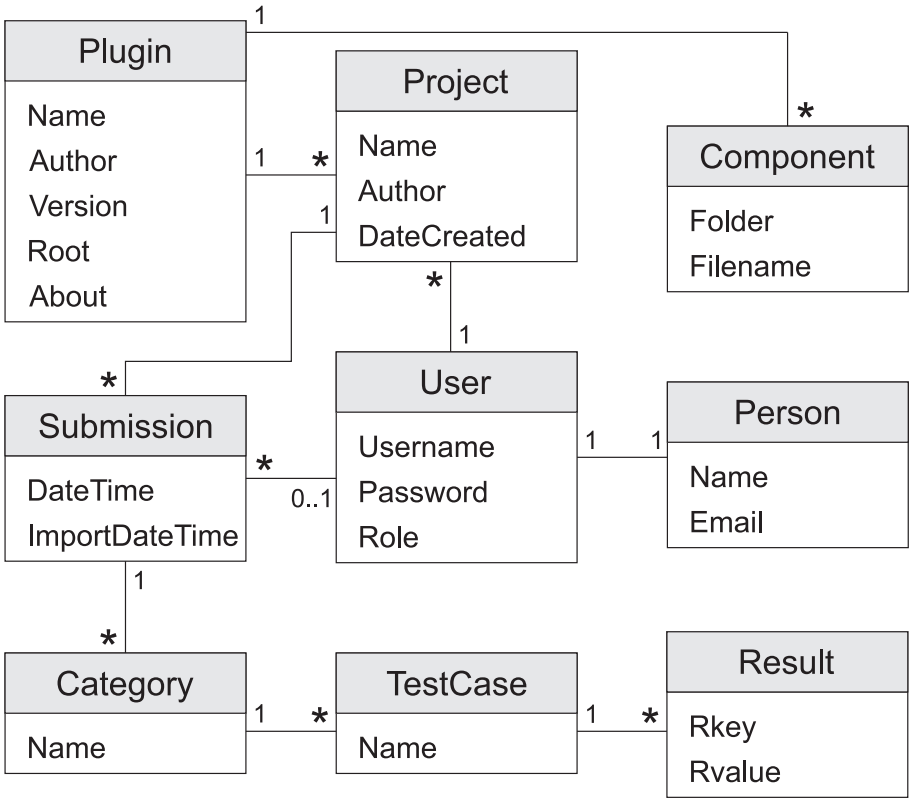
Obrázek 3.2: Schéma aplikace

Jádro aplikace Řízení datových toků má na starosti aplikační jádro, které umožňuje integraci neomezeného množství zásuvných modulů. Zprostředkovává příjem dat výsledkových sad a předává data prezentační vrstvě pro vizualizaci. Při importu data dává příslušnému zásuvnému modulu pro zpracování a poté je předává datové vrstvě pro uložení. Při předávání dat prezentační vrstvě jsou nejdříve zpracována zásuvným modulem.

Zásuvný modul Role zásuvného modulu spočívá ve zpracování dat daného formátu testovací sady při importu a při jejich vizualizaci. Pro vizualizaci je zásuvnému modulu umožněno použít vestavěných komponent, nebo implementovat vlastní. Tento mechanizmus je popsán v podsekci 4.2.3.

3.3 Datový model

Datový model aplikace musí odpovídat obecnému vzoru testovacích sad, který vychází ze specifikace v kapitole 2. Testovací sady se skládají z testovacích případů, které mohou být rozděleny do kategorií. Tyto testovací případy pak obsahují žádný či více výsledků. Výsledkové sady je potřeba ukládat odděleně, aby bylo možné zobrazit jejich detail a umožnit jejich vzájemné porovnání. Dále je nutné umožnit rozdělení do jednotlivých projektů, kdy je jeden formát testovací sady využíván pro různé testovací cykly.



Obrázek 3.3: Diagram tříd

Na obrázku 3.3 je zobrazen návrh datového modelu pomocí diagramu tříd. Tento model dále obsahuje entity, které umožňují správu uživatelů.

Plugin Entita *Plugin* představuje instalovaný zásuvný modul aplikace.

Project Každý jednotlivý zásuvný modul může obsahovat více samostatných projektů, které sdílí způsob zpracování, avšak představují oddělené celky.

Submission Entita *Submission* představuje jednu výsledkovou sadu. Obsahuje jednoznačný identifikátor, pomocí kterého lze jednotlivé výsledkové sady rozlišit jak programově, tak uživatelsky.

Category Umožnuje členění výsledkové sady do kategorií.

TestCase Představuje testovací případy testovací sady. Může obsahovat jeden či více výsledků daného testovacího případu definovaných entitou *Result*.

Result Výsledek testovacího případu, který nese klíč, jako své označení, a výslednou hodnotu.

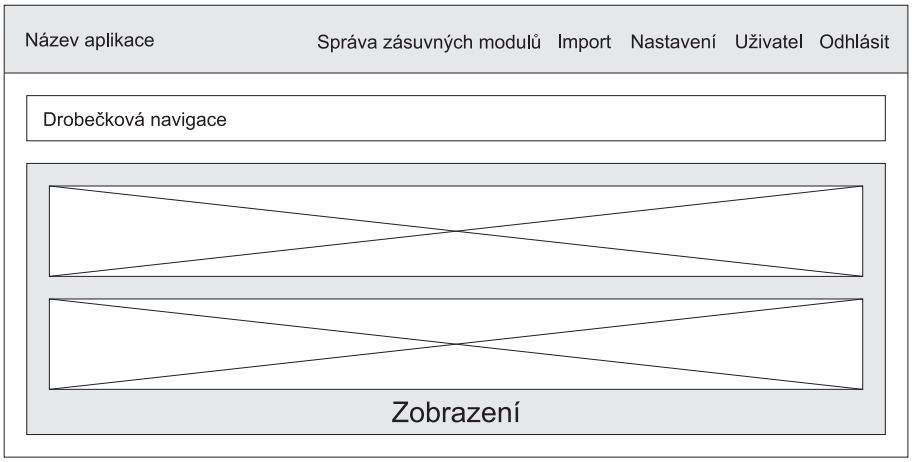
User Informace o uživateli, sloužící pro jeho jednoznačnou identifikaci a přihlášení do systému.

Person Doplňující informace o uživateli, které slouží jako kontaktní údaje.

Component Tato entita slouží pro možnost vkládání komponent definovaných zásuvnými moduly, jejich jednodušší správu a zavedení do uživatelského rozhraní aplikace.

3.4 Uživatelské rozhraní

Aby bylo uživatelské rozhraní co nejintuitivnější, byla aplikace navržena s co nejnižším množstvím navigačních prvků, které dále odkazují na veškerou nutnou funkcionalitu. Pro větší bezpečnost, respektive řízení přístupu k datům, je aplikace přístupná pouze přihlášeným uživatelům. Návrh aplikace s rozložením navigačních prvků je na obrázku 3.4.



Obrázek 3.4: Schéma aplikace

Aplikace je rozdělena na čtyři hlavní sekce. První sekce, Správa zásuvných modulů, odkazuje na primární funkcionalitu aplikace, tedy prezentaci dat testovacích sad. Pro import nových výsledkových sad slouží sekce Import. Nastavení nabízí možnosti přizpůsobení, jako například správu uživatelů. Sekce Uživatel pak umožňuje zobrazení a editaci profilu přihlášeného uživatele.

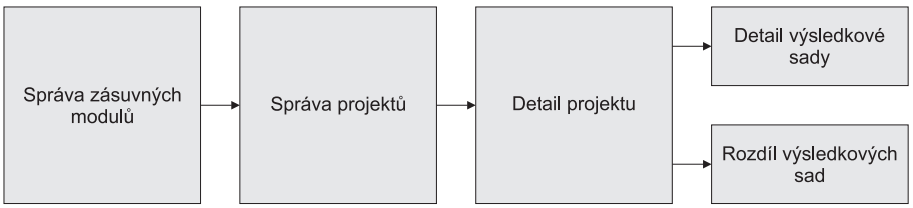
Hlavní navigační lišta je neměnná a slouží pouze pro zobrazení zmíněných sekcí. Ovládací prvky jednotlivých zobrazení jsou součástí zobrazených dat. Ke zjednodušení práce s aplikací byla přidána drobečková navigace, která vytváří cestu k aktuálnímu zobrazení a zároveň poskytuje zpětné odkazy. Tento ovládací prvek byl zvolen také kvůli záměru přizpůsobit aplikaci mobilním zařízením.

Se stále narůstajícím podílem mobilních zařízení, skrze které uživatelé navštěvují weby, rostou i požadavky uživatelů na responzivní aplikace. Tyto aplikace se vyznačují přizpůsobením obsahu a rozložením prvků pro malé a střední obrazovky mobilních zařízení a umožňují tedy snadnější práci i na těchto typech zařízení. Z tohoto důvodu je aplikace od začátku navrhována s důrazem na responzivitu.

3.4.1 Správa zásuvných modulů

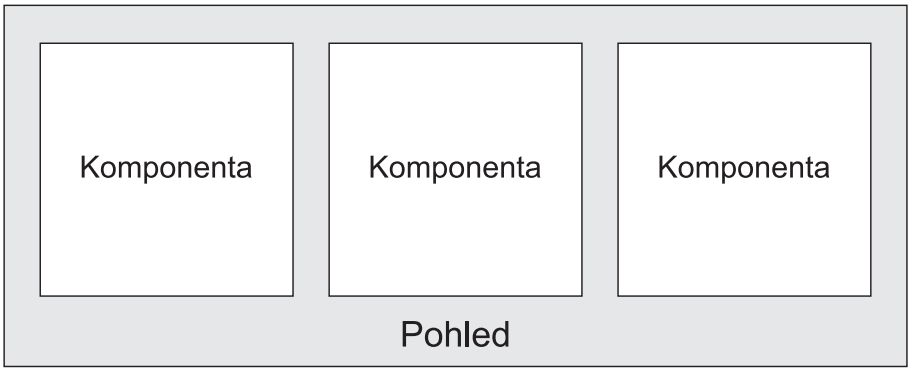
Pro zobrazení dat testovacích sad slouží správa zásuvných modulů. Úvodní obrazovka obsahuje výpis nainstalovaných zásuvných modulů, respektive dostupných formátů testovacích

sad, které je aplikace schopna zpracovávat. Po výběru zásuvného modulu se uživatel dostává na výpis projektů. Následující pohledy jsou navrženy tak, aby je bylo umožněno dynamicky modifikovat pomocí implementace zásuvného modulu. Jedná se o detail projektu, detail výsledkové sady a rozdíl výsledkových sad. Navigace správou zásuvných modulů je na obrázku 3.5.



Obrázek 3.5: Navigace správou zásuvných modulů

Dynamické pohledy Mezi dynamické pohledy patří detail projektu, detail výsledkové sady a rozdíl výsledkových sad. Tyto pohledy se skládají z komponent, které jsou na sobě nezávislé a zobrazují data výsledkové sady ve formátu, jakém je definuje implementace dané komponenty konkrétního zásuvného modulu. Tento návrh umožňuje použití neomezeného počtu komponent v každém z dynamických pohledů, a zároveň umožňuje vlastní implementaci komponent zásuvným modulům. Schéma tohoto návrhu je na obrázku 3.6.



Obrázek 3.6: Schéma dynamického pohledu

Detail projektu Tento výpis má za úkol výpis výsledkových sad pro jednoduchou navigaci mezi detaily výsledkových sad, či volby pro jejich rozdíl. Zároveň sem lze umístit komponenty, které pracují s daty nad více výsledkovými sadami, mezi které se může řadit například souhrnný graf výsledků.

Detail výsledkové sady Detail výsledkové sady pracuje s daty konkrétní výsledkové sady. Patří sem výpis testovacích případů dělených do kategorií včetně jejich výsledků. Při formátu testovací sady, kde je umožněno více výsledků, se nabízí zobrazení výsledků seskupených podle testovacího případu. Mezi další komponenty lze zařadit takové, které provádí nad výsledky agregační funkce, nebo komponenty s detailem konfigurace, na které testování proběhlo.

Rozdíl výsledkových sad Pro širší využití je umožněno porovnání dvou a více testovacích sad. V tomto pohledu jsou navrženy komponenty, které barevným zabarvením

buňky tabulky detekují přechod do rozdílného výsledku testovacího případu. Dále pak porovnání agregátů testovacích sad, či rozdíly v použitých konfiguracích.

Kapitola 4

Implementace aplikace

Tato kapitola popisuje konkrétní řešení problémů, týkajících se aplikace a její komunikace se zásuvnými moduly, včetně návaznosti na použité nástroje.

4.1 Datová vrstva

Datový model aplikace musí odpovídat obecnému vzoru testovacích sad, přičemž je kromě samotného ukládání výsledných dat nutné tato data doplnit o údaje pro identifikaci nejen v rámci konkrétního zásuvného modulu, ale i v rámci celé aplikace, aby tato data byla od sebe oddělena. Datový model je popsán v sekci 3.3. Zde jsou popsány entity, u kterých je potřeba doplnit implementační detaily.

Plugin Entita *Plugin* obsahuje základní informace o zásuvném modulu a umožnuje jádru identifikovat tento modul na aplikační vrstvě pomocí názvu kořenového adresáře, v tomto případě atribut *Root*, a tím volat rozhraní příslušného zásuvného modulu. Kořenovým adresářem je myšleno umístění všech instalovaných zásuvných modulů aplikace, čímž je adresář *plugins*.

Submission Název každé takovéto entity je převzat z času proběhnutí testování, a to ve formátu, ve kterém je ve výsledkové sadě uvedeno. Pokud datum uvedeno ve výsledku není, je umožněno toto datum vložit při importu dat, nebo naplnit identifikátorem, který vytvoří samotný zásuvný modul.

Entity modulu Kromě implementace datového modelu aplikace je umožněno zásuvným modulům doplnit tento model o vlastní entitu, do které jim je povolen přístup při importu dat a při získávání dat pro vizualizaci.

4.1.1 MySQL

Pro implementaci datového modelu je použit multiplatformní databázový systém MySQL, který se v dnešní době řadí mezi nejpoužívanější. Tento systém byl zvolen z důvodu ověřené a funkční kombinace s jazykem PHP a také kvůli jeho otevřenosti, respektive dostupnosti.

4.2 Prezentační vrstva

Jedná se o nejkomplexnější vrstvu aplikace. Je tvořena základním uživatelským prostředím a komponenty pro zobrazení výstupních dat.

4.2.1 Uživatelské rozhraní

Pro uživatelskou interakci se systémem slouží uživatelské rozhraní postavené na nadstavbě jazyka JavaScript, AngularJS, podporované rozhraním Foundation. Jeho úkolem je nejen správa aplikace jako takové, ale i správa zásuvných modulů a navigace mezi těmito moduly na uživatelské úrovni.

AngularJS Tato nadstavba jazyka JavaScript umožnuje vytvářet dynamické webové aplikace bez nutnosti duplicitního načítání její struktury. Kontakt s aplikační vrstvou je zejména na úrovni datové, což snižuje zátěž této vrstvy a šetří celkový výpočetní výkon, který je částečně přenesen na vrstvu prezentační. Veškeré potřebné informace o tomto rozhraní byly získávány z dokumentace na oficiálních webových stránkách [4].

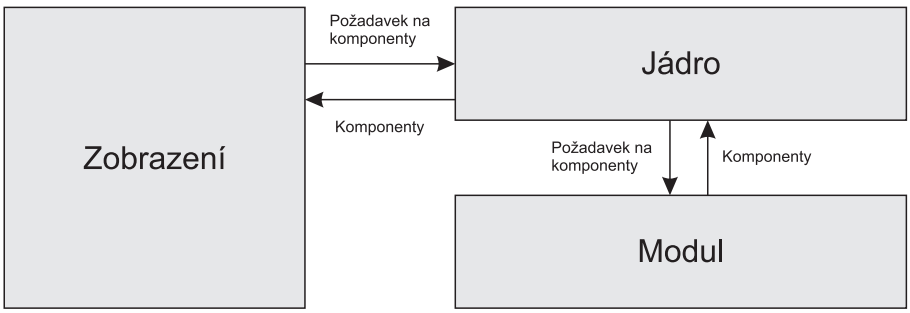
Foundation Pro rozšířené použití aplikace i na mobilních zařízeních je použito CSS rozhraní Foundation. Toto rozhraní umožnuje poměrně rychlý vývoj webových aplikací s nativní podporou mobilních zařízení, ať už se jedná o mobilní telefony, či tablety. Výstupem tohoto rozhraní je tedy responzivní aplikace. Rozhraní je detailně zdokumentováno na oficiálních stránkách [11].

4.2.2 Import dat

Import dat byl implementován částečně pomocí prezentační vrstvy, skrze kterou přijímá data od uživatele ve formě textových souborů. Tato data jsou uživatelem doplněna o formát zásuvného modulu a příslušný projekt a následně nahrána na aplikační vrstvu aplikace, kde dochází k dalšímu zpracování, které je popsáno v kapitole 4.3.3.

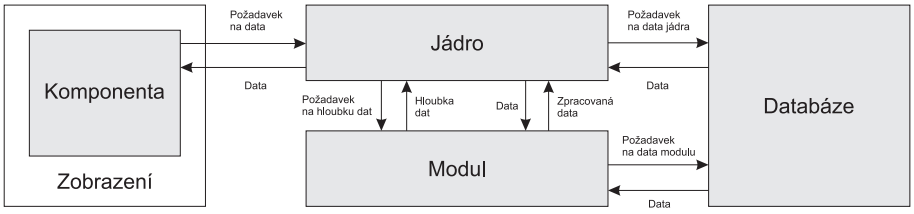
4.2.3 Zobrazení dat

Zobrazení dat je rozděleno na tři samostatné pohledy. Jedná se o detail projektu, detail testovací sady a detail porovnání testovacích sad. Popis těchto pohledů je v kapitole 3.4.1. Každé toto zobrazení implementuje dynamické prostředí, jehož komponenty jsou dány implementací konkrétního zásuvného modulu. Schéma této implementace je zobrazeno na obrázku 4.1.



Obrázek 4.1: Schéma komunikace při žádosti o komponenty

Při načtení pohledu dojde k zaslání požadavku na aplikační vrstvu, kde jádro zažádá o data od zásuvného modulu. Tato data obsahují identifikaci komponent pro zobrazení. Prezentační vrstva tyto komponenty zakomponuje do modelu DOM a naváže na prostředí AngularJS. Následně si komunikaci s aplikační vrstvou řídí každá komponenta sama. Tato komunikace byla ze strany jádra implementována jako asynchronní.



Obrázek 4.2: Schéma komunikace komponenty s aplikační vrstvou

Jak lze vidět na obrázku 4.2, komponenta zasílá požadavek na aplikační vrstvu, který obsahuje její identifikátor. Jádro se nejprve dotáže příslušného modulu na hloubku dat, která jsou potřeba pro danou komponentu, a pak tato data získá z databáze. Hloubkou dat je zde myšlena úroveň entitní závislosti. Minimální hloubka odpovídá výsledkové sadě, maximální hloubka pak konkrétním výsledkům. Díky tomuto se snižuje zátěž a potřebný čas pro zpracování dat.

Po získání dat jádro předává tato data ke zpracování modulu, který je obalí do objektů pro danou komponentu. Během tohoto procesu může modul využít data z vlastní tabulky v databázi. Zpracovaná data jsou předána zpět jádru a následně komponentě, která požadavek na data zaslala. Komponenta pak tato data na základě své implementace zobrazí.

Direktiva AngularJS O zobrazení konkrétní datové scény se stará komponenta nadstavby jazyka JavaScript, AngularJS, zvaná direktiva. Tato komponenta umožňuje tvorbu vlastních struktur, které lze dynamicky vkládat do obsahu aplikace, a tím zajistit již zmíněnou modularitu. Tato komponenta je základem pro tvorbu komponent zásuvných modulů pro zobrazení dat.

4.3 Aplikační vrstva

Aplikační vrstva zajištuje komunikaci mezi prezentační a datovou vrstvou. Dále vytváří prostředí pro integraci zásuvných modulů, kterým zprostředkovává komunikaci se zmíněnými vrstvami.

4.3.1 Implementace aplikační vrstvy

Aplikační vrstva byla implementována pomocí skriptovacího jazyka PHP [5]. Bylo tak zvoleno z důvodu vlastní zkušenosti s tímto jazykem, a zároveň z důvodu rozšířit si v oblasti tohoto jazyka své znalosti. Tento jazyk vznikl především pro implementaci dynamických webových stránek ze strany serveru.

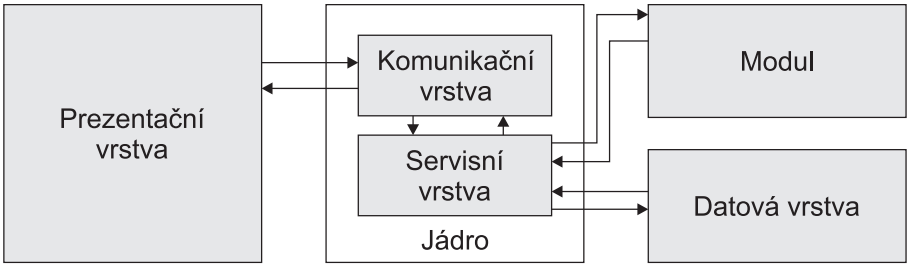
Během implementace bylo dbáno na přehledný a čistý kód, doplněný o množství komentářů. Dále bylo využito čistě objektového přístupu, což umožňuje lépe strukturovat kód do sémantických celků. Každá třída je umístěna do samostatného souboru.

I přes prvotní účel vzniku jazyka PHP je v aplikaci použit pouze na aplikační vrstvě. Dynamičnost grafického výstupu je zajišťována prezentační vrstvou.

4.3.2 Jádro

Výchozí část aplikační vrstvy, nazvaná jádro, spravuje chod celé aplikace a jednotlivé zásuvné moduly. Dále řídí přístupy zásuvných modulů k jednotlivým vrstvám a podle identifikace požadavků volí příslušné nástroje, či data, která jsou od zásuvného modulu z důvodu bezpečnosti odstíněna.

Samotná implementace jádra je rozdělena na dvě části. První část představuje komunikační vrstvu, druhá vrstvu servisní. Komunikační vrstva přebírá požadavky prezentační vrstvy a volá příslušné metody vrstvy servisní, která výsledky předává zpět pro odeslání prezentační vrstvě.



Obrázek 4.3: Schéma komunikace v rámci jádra

Součástí jádra je i implementace několika podpůrných nástrojů, které vznikly během vývoje aplikace:

dbCreator Tento nástroj představuje objektový přístup ke tvorbě databázové struktury. Atributy a jednotlivé entity jsou reprezentovány objekty, které se následně pomocí speciálních metod transformují na databázové dotazy. Ukázka na obrázku 4.4 inicializuje jednoduchou tabulku Uživatel.

```
// Inicializuj tabulku User
$tUser = new DbTable('User');

// Nastav povinny atribut Id
$pId = new DbProperty('Id');
$pId->SetType(DbType::Double());
$pId->NotNull();
// Nastav priznak primarniho klice
$pId->PrimaryKey();
// Nastav automaticke zvysovani hodnoty
$pId->AutoIncrement();

// Pridej atribut do tabulky
$tUser->AddProperty($pId);
...
// Ziskej dotaz pro vytvoreni tabulky
$tUser->GetTableDefinition();
```

Obrázek 4.4: Inicializace tabulky Uživatel nástrojem dbCreator

CorlyADL Pro komunikaci s databází byla vytvořena abstraktní databázová vrstva, která se záznamy v databázi pracuje jako s objekty. Toto umožnilo automatizaci práce s databází a tudíž rychlejší implementaci.

Základem této vrstvy je třída implementující připojení k databázi a dynamickou tvorbu databázových dotazů. Pro každou entitu je vytvořena zděděná třída *Dao*, která zastřešuje komunikaci s danou entitou v databázi. S touto entitou pak lze pracovat pomocí několika metod:

Save uložení objektu do databáze,

Load načtení objektu z databáze podle Id,

GetList seznam objektů dané entity,

GetFilteredList filtrovaný seznam dane entity.

LINQ K jednodušší práci s poli objektů slouží nástroj LINQ, což je v tomto případě obdoba nástroje z jazyka C#. LINQ je dotazovacím jazykem, sloužícím k získání potřebných dat. V aplikaci obaluje standardní implementaci pole, čímž umožňuje vytváření složitějších, ale zároveň čitelnějších dotazů nad polem objektů. Na obrázku 4.5 je ukázka použití tohoto nástroje pro získání seřazeného seznamu uživatelských jmen z pole uživatelů.

```
// Ziskej serazeny seznam uzivatelskych jmen z pole uzivatelu,
// kde neni zadano heslo
$lUsers = new LINQ($users);
$usernamesWithoutPassword = $lUsers
    ->Where('Password', LINQ::IS_EQUAL, '')
    ->OrderBy('Username')
    ->Select('Username')
    ->ToList();
```

Obrázek 4.5: Ukázka dotazu na pole výsledků pomocí nástroje LINQ

ValidationResult Pro jednotnou validaci napříč aplikací vznikl nástroj *ValidationResult*. Tento nástroj obaluje validovaný objekt a umožňuje nad ním provádět vestavěné validační kontroly. Výsledek validace je pak nastavován speciálnímu příznaku a do pole jsou ukládány chybové hlášky pro další zpracování prezentační vrstvou. Ukázka jednoduché validace, zda bylo zadáno heslo, je na obrázku 4.6.

```
// Inicializuj validaci a zkontroluj,
// zda ma uzivatel zadane heslo
$validation = new ValidationResult($credentials);
$validation->CheckNotNullOrEmpty('Password',
'Heslo musi byt zadano');
// Pokud ne, ukonci a vrat validacni objekt
if (!$validation->IsValid)
    return $validation;
```

Obrázek 4.6: Ukázka validační kontroly pomocí nástroje ValidationResult

TSE Spíše než nástroj je TSE sada objektů pro práci s daty testovacích sad. Každá entita testovací sady má svou TSE (Test-Suite-Entity), která tuto entitu obaluje. Díky tomuto je zajištěna jednotná práce s tímto typem dat v komunikaci mezi modulem a jádrem aplikace.

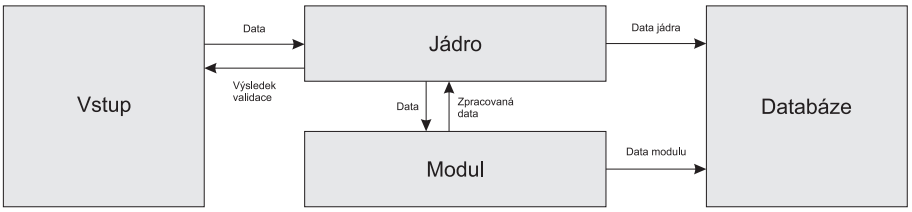
Kromě zajištění jednotnosti jsou objekty vybaveny podpůrnými metodami, jako například mapování objektu pro uložení do databáze a zpět, či export objektu pro serializaci.

Komponenty Pro základní vizualizaci jádro nabízí několik komponent, které jsou využitelné pro všechny moduly. Jedná se například o výpis výsledkových sad projektu, graf, nebo modifikovatelný výpis testovacích případů. Všechny tyto komponenty jsou nastavitelné ze strany modulu.

4.3.3 Zásuvný modul

Zásuvný modul jako takový zajišťuje funkcionalitu nad konkrétním typem testovací sady. Pro správnou integraci s jádrem aplikace musí implementovat dvě komunikační rozhraní a disponovat vlastním konfiguračním souborem.

Importer *Importer* je rozhraní pro import dat, které je voláno jádrem při vkládání nových výsledků testovacích sad. Účelem tohoto rozhraní je zpracování a validace vstupních dat, jejich obalení do databázových objektů a následné předání zpět jádru, které tyto data ukládá do databáze. Během tohoto procesu je modulu umožněn přístup k vlastní tabulce. Schéma této výměny dat je zobrazeno na obrázku 4.7.



Obrázek 4.7: Schéma importu dat

Visualizer Pro zobrazení výsledných dat slouží rozhraní *Visualizer*. Toto rozhraní definuje metody pro získání komponent jednotlivých pohledů a metody pro získání dat, které se těmto komponentám předávají. Tato komunikace je popsána v kapitole 4 tohoto textu.

Entita Entita je nepovinnou částí zásuvného modulu, ke které může přistupovat během importu dat, nebo při zpracování dat pro konkrétní komponentu. Jedná se o rozšíření datového modelu aplikace, kam si může zásuvný modul ukládat svá vlastní data.

Komponenty Další nepovinou součástí modulu je jedna či více komponent. Každá takováto komponenta může být poté využívána prezentační vrstvou pro zobrazování dat daného modulu. Jak již bylo zmíněno v kapitole 4, komponenty jsou implementovány pomocí direktiv. Tyto direktivy představují oddělenou část prezentační vrstvy, která je plněna odpovídajícími objekty vrstvy aplikační.

Konfigurační soubor Konfigurační soubor, ve tvaru XML, slouží pro instalaci zásuvného modulu. Jeho tvar si lze prohlédnout na obrázku 4.8.

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
    // Zakladni informace o modulu
    <base>
        <name>TEXT</name>
        <author>TEXT</author>
        <version>TEXT</version>
        <root>TEXT</root>
        <about>TEXT</about>
    </base>
    // Entita modulu
    <entity name="eName">
        <property name="pName" type="varchar" value="63" primary-key="true"
        auto-increment="false"></property>
    </entity>
    <components>
        <component folder="fName">
            <file>TEXT</file>
        </component>
    </components>
</config>
```

Obrázek 4.8: Ukázka konfiguračního souboru pro instalaci zásuvného modulu

Skládá se ze tří částí. První část, *base*, definuje základní informace o modulu, jako například jeho název či autora. Druhá část, *entity*, definuje vlastní entitu modulu. Ta může být maximálně jedna. Poslední, nepovinná část, nazvaná *components*, definuje komponenty pro vizualizaci výsledků. Počet těchto komponent není jádrem nijak omezován.

Kapitola 5

Případové studie

V rámci této bakalářské práce byli instalovány tři zásuvné moduly, které implementují veškerou funkcionalitu popsanou v kapitole 4. Tyto moduly využívají jak komponenty poskytnuté jádrem, tak komponenty vlastní, které slouží pro demonstraci nabízených možností samotné aplikace, nejedná se tedy o plnohodnotné zásuvné moduly. V této kapitole jsou rozebrány případové studie dvou z nich.

5.1 Zásuvný modul GLIBC Make v1.0

Tento zásuvný modul implementuje zpracování výstupu vzniklého spuštěním testovací sady zabudované přímo v Makefile knihovny GNU C. Testovací sada se spouští pomocí příkazu `make xcheck`. Výstupní soubor obsahuje množství trasovacích informací, které vytváří jeden celistvý a nepřehledný shluk dat. Informace o tomto formátu testovací sady byly převzaty z komunitních wiki stránek projektu [7].

5.1.1 Vstupní data

V ukázce 5.1 je zobrazen výtažek vstupních dat.

```
...
gcc -nostdlib -nostartfiles -o /root/rpmbbuild/BUILD/glibc-2.12-2-
gc4ccff1/build-x86_64-linuxnptl/posix/tst-waitid -Wl,-dynamic-linker...
make[2]: *** [posix/tests] Error 2
...
```

Obrázek 5.1: Ukázka vstupních dat pro zásuvný modul GLIBC Make v1.0

Implementace modulu se zaměřuje pouze na chyby. Ty mají ve výstupu následující tvar:

```
make[2]: *** [posix/tests] Error 2
```

Tuto chybu lze rozčlenit na několik částí, které lze přiřadit obecnému tvaru výsledků testovacích sad:

Kategorie Číselné označení *make*

Testovací případ Označuje případ, kdy k dané chybě došlo. Tuto hodnotu lze získat z řetězce mezi hranatými závorkami

Výsledek Jako výsledek je v tomto modulu považován kód chyby, přičemž klíč výsledku je stále stejný, a tím je řetězec *Error*. Datum proběhnutí překladu není v tomto případě uveden, je tedy modulem nastavován na aktuální čas zpracování dat, což nemusí být žádoucí. Řešením tohoto problému může být umožnit uživateli zadat čas při importu dat, nebo získávat identifikátor jiného než časového formátu.

5.1.2 Očekávaný výstup

Z výše zmíněných dat lze získat několik poznatků, rozdělených do jednotlivých pohledů

Detail projektu

- souhrnný výpis testovacích sad
- přehled chybových kódů a jejich počtů napříč projektem

Detail testovací sady

- přehled chybových kódů a jejich počtů v testovací sadě
- výpis případů v rámci kategorií s chybovými kódy

Detail porovnání testovacích sad

- přehled chybových kódů a jejich počtů napříč porovnávaných testovacích sad
- rozdíly v počtu chyb
- rozdíly v počtu chyb v rámci kategorií
- výpis případů se zvýrazněnými přechody do stavu s chybou či bez chyby

5.1.3 Výstupy aplikace

Tento modul implementuje veškerá rozhraní, která jsou popsána v kapitole 4.

Import dat Import dat je prováděn z textového souboru. Jednotlivé testovací případy jsou vyhledány pomocí regulárního výrazu.

Vizualizace dat Data jsou vizualizována tak, aby poskytovala informace, které jsou popsané v podsekci Očekávaný výstup. K tomuto účelu bylo implementováno několik komponent:

- rozdíly v počtu chyb výsledkové sady
- rozdíly v počtu chyb v rámci kategorií

5.2 Zásuvný modul SystemTAP v1.0

SystemTap je diagnostický nástroj, který slouží k získávání informací o běžícím systému Linux. Definuje skriptovací jazyk *stap*, pomocí kterého jsou vytvářeny sondy pro zachycení určitých událostí, a zároveň kroky, které se po zachycení těchto událostí mají provést. Výstupem těchto kroků jsou rozsáhlá textová data, která mohou obsahovat tisíce výsledků, ale i zároveň dat, která nejsou pro zpracování podstatná. Jedná se tedy o poměrně datově náročný formát testovací sady, co se velikosti týče.

5.2.1 Vzor vstupních dat

Na obrázku 5.2 je uveden výtažek výstupních dat tohoto nástroje.

```
Test Run By root on Mon Aug 19 13:43:13 2013
Native configuration is x86_64-unknown-linux-gnu

    === systemtap tests ===

Running
...

Host: Linux pes-guest-104.lab.eng.brg.redhat.com 3.10....
Snapshot: version 2.2.1/0.155, rpm 2.2.1-1.e17
GCC: 4.8.1 [gcc (GCC) 4.8.1 20130612 (Red Hat 4.8.1-2)]
Distro: Red Hat Enterprise Linux Server release 7.0 Beta (Maipo)
SElinux: Enforcing
...

Running ./systemtap.apps/java.exp ...
UNTESTED: java - no javac
Running ./systemtap.base/add.exp ...
executing: stap -v ./systemtap.base/add.stp
spawn stap -v ./systemtap.base/add.stp
Pass 1: parsed user script and 93 library script(s)
using 105088virt/26356res/2828shr/23952data kb, in 130usr/10sys/158real ms.
Pass 2: analyzed script: 2 probe(s), 0 function(s),0 embed(s), 3 global(s)
using 105616virt/26884res/2892shr/24480data kb, in 10usr/0sys/4real ms.
Pass 3: translated to C into "/tmp/stapNYPFRF/stap_99a362b2c56c7eae8a5555
bd18acdb1c_1090_src.c" using 105616virt/27252res/3196shr/24480data kb, in
 0usr/0sys/0real ms.
Pass 4: starting run.
systemtap starting probe
PASS: ./systemtap.base/add.stp startup
PASS: ./systemtap.base/add.stp load generation
...
```

Obrázek 5.2: Ukázka vstupních dat pro zásuvný modul SystemTAP v1.0

Ze vzorku dat lze vyčíst základní informace, které modul z výstupu testovací sady získává. První informací je datum, kdy k testu došlo, další konfigurace, na které probíhal. Testovací případ má v tomto modulu následující tvar:

```
Running ./systemtap.apps/java.exp ...
UNTESTED: java - no javac
```

Tento testovací případ lze rozdělit do obecných entit:

Kategorie Zásuvný modul pro zjednodušení nepočítá s rozdělením do kategorií. Je proto počítáno s jednou, pojmenovanou *Výchozí*. Výstup testovací sady však lze dělit do kategorií podle přípony výrazu *systemtap* na řádku s *Running*.

Testovací případ Testovacím případem je zde spuštění souboru, jehož název se nachází za slovem *Running*.

Výsledek Vše, co následuje po spuštění souboru a splňuje formát výsledku. Tento formát je definován jako stav uvedený před znakem dvojtečky, což představuje hodnotu, a klíčem, což je vše, co se nachází za zmíněným znakem. Množina klíčů, která je zpracovávána zásuvným modulem, je převzata z nástroje Dejazilla [3]. Těmito klíči jsou: PASS, FAIL, KPASS, KFAIL, XPASS, XFAIL, UNTESTED, UNRESOLVED, UNSUPPORTED, WARNING a ERROR.

5.2.2 Očekávaný výstup

Z výše uvedeného formátu dat lze získat několik poznatků, rozdělených do jednotlivých pohledů.

Detail projektu

- souhrnný výpis testovacích sad
- souhrnný přehled výsledků napříč testovacích sad

Detail testovací sady

- konfigurace, na které testování proběhlo
- přehled výsledků napříč testovací sadou
- výpis testovacích případů včetně výsledků s barevným podbarvením podle charakteru výsledku (zelené odstíny pro kladné výsledky, červené pro záporné)

Detail porovnání testovacích sad

- přehled konfigurací
- výpis případů se zvýrazněnými přechody stavů

5.2.3 Výstupy aplikace

Modul pro zpracování dat z nástroje SystemTap implementuje rozhraní, která jsou popsána v kapitole 4.

Import dat Data, která jsou přijata v textovém souboru, jsou zpracovávána po řádku. Každý takovýto řádek je pak kontrolován regulárním výrazem, jelikož je nejprve potřeba dojít k testovacímu případu, a následně k tomuto případu přiřazovat jednotlivé výsledky.

Kromě testovacích případů je z importovaných dat získána i konfigurace, na které nástroj běžel, která je následně k dané výsledkové sadě uložena do vlastní tabulky modulu.

Vizualizace dat Vizualizace dat odpovídá popisu z podsekce Očekávaný výstup. K tomuto účelu bylo implementováno několik komponent, včetně takové, která získává data uložená v tabulce modulu. Mezi tyto komponenty patří:

- konfigurace, na které testování proběhlo
- přehled konfigurací

5.3 Zásuvný modul CPAlien Forester v1.0

Mimo zásuvných modulů GLIBC Make v1.0 a SystemTAP v1.0 byl také implementován zásuvný modul CPAlien Forester v1.0. Tento modul zpracovává výsledky testovací sady zásuvného modulu Forester nástroje CPAlien, implementovaného pomocí rozhraní CPAChecker. Testovací sada slouží pro verifikaci programů, které manipulují s komplexními dynamickými datovými strukturami [6].

Výstupem jednoho spuštění testovací sady je několik souborů typu xml, které představují samostatné kategorie. Implementovaný zásuvný modul však pro zjednodušení a z důvodu nedostatku testovacích dat zpracovává vždy pouze jeden výstupní xml soubor, který považuje za výsledek jednoho spuštění testovací sady. Testovací případy sdílejí pevný počet výsledků stejného typu, což zjednodušuje výsledné zpracování zásuvným modulem. V rámci tohoto zásuvného modulu byla implementována komponenta pro zobrazení maximálních a minimálních hodnot nad výsledky čas strávený procesorem, použitá paměť a reálná doba zpracování.

5.4 Vyhodnocení

Implementace aplikace umožňuje fungující integraci zásuvných modulů za předpokladu, že tyto dodrží aplikační rozhraní popsané v kapitole 4. Zároveň poskytuje rozšiřitelné uživatelské rozhraní z pohledu zobrazení výsledkových sad různých formátů pomocí komponent, které umožňují zobrazit data prakticky v jakékoliv podobě. Zmíněnou integraci a zobrazení dat považuji za největší přínos implementovaného nástroje.

Během testování se aplikace potýkala se dvěma zásadními problémy. Prvním problémem je import dat, který probíhá za účasti uživatele, který musí ze souborového systému zvolit textový soubor pro import. Pokud se jedná o soubor o velkém počtu dat, stává se aplikace po dobu nahrávání na aplikační vrstvu pro daného uživatele nepoužitelnou, což zpomaluje celkovou práci s aplikací. Dalším problémem je výkon aplikace při načítání dat z databáze. Při velkém počtu dat dochází k prodlevě až několik desítek sekund. Tato prodleva byla

zčásti vyřešena na aplikační vrstvě pomocí asynchronního volání jednotlivých komponent a určováním hloubky dat, avšak hlavním problémem je abstraktní databázová vrstva, která svou implementací neumožňuje složitější dotazy.

Kapitola 6

Závěr

Tato bakalářská práce se zabývala návrhem a implementací webového nástroje pro správu výsledků dlouhodobého testování. Cílem tedy bylo vytvořit aplikaci schopnou zpracování rozsáhlých výsledků testovacích sad. Pro širší využití aplikace však bylo vyžadováno, aby aplikace bylo schopná zpracovávat výstupy testovacích sad různých formátů.

Výsledná aplikace implementuje jádro, které je možno rozšiřovat o zásuvné moduly, které zajišťují zpracování dat konkrétního formátu testovací sady. Tento zásuvný modul také implementuje komponenty pro jejich zobrazení. Samotné jádro vytváří aplikační prostředí, zajišťuje interní správu dat a zásuvným modulům poskytuje komunikační rozhraní včetně podpůrných nástrojů.

Další vývoj aplikace je směrován k rozšíření aplikačního jádra. Příkladem může být například rozšíření o více možností importu dat, optimalizace komunikace s databází včetně podpory rekurzivních dotazů a rozšíření aplikace o uživatelská nastavení ulehčující práci s aplikací, či konfiguraci jednotlivých projektů, kde se nabízí možnost výběru komponent pro vlastní vizualizaci.

Literatura

[1] *Automated Detection of Performance Regressions: The Mono Experience*, In proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005), 2005, ISBN 0-7695-2458-3, ISSN 1526-7539.

[2] SnapshotTesting - Systemtap Wiki [online]. https://sourceware.org/systemtap/wiki/SnapshotTesting, 2007 [cit.2014-05-18].

[3] Dejazilla [online]. https://web.elastic.org/ dejazilla/view.php, [cit.2014-05-18].

[4] Google: AngularJS API Docs [online]. https://docs.angularjs.org/api, 2014 [cit.2014-05-18].

[5] Group, T. P.: PHP: Documentation [online]. http://www.php.net/docs.php, 2014 [cit.2014-05-18].

[6] HABERMEHL, P.; HOLIK, L.; ROGALEWICZ, A.; aj.: Forester - Tool for Verification of Programs with Pointers [online]. http://www.fit.vutbr.cz/research/groups/verifit/tools/forester/cs, 2013 [cit.2014-05-19].

[7] O'DONNEL, C.: glibc wiki: Testing/Testsuite [online]. https://sourceware.org/glibc/wiki/Testing/Testsuite, 2013 [cit.2014-05-18].

[8] Red Hat: Generating Instrumentation for Other Computers [online]. https://access.redhat.com/site/documentation/enUS/Red_Hat_Enterprise_Linux/5/html/SystemTap_Beginners_Guide/cross-compiling.html, 2014 [cit.2014-05-18].

[9] SAVOYE, R.; ELISSTON, B.: DejaGnu: The GNU Testing Framework [online]. http://www.gnu.org/software/dejagnu, 2011 [cit.2014-05-18].

[10] WIELAARD, M.: A SystemTap update [online]. http://lwn.net/Articles/315022, 2009 [cit.2014-05-18].

[11] Zurb Inc.: Gettings Started — Foundation Docs [online]. http://foundation.zurb.com/docs/, 2014 [cit.2014-05-18].

Příloha A

Obsah CD

- zpráva ve formátu PDF
- zdrojový tvar písemné zprávy
- návod k instalaci
- zdrojové texty
- zkušební data testovacích sad SystemTap, glibc a CPAlien Forester